Instructions for running/testing our code:

Note on discounts and tax: we made the design decision to not allow these to be set by the user, but be specified in the central database, and automatically applied in the apps. This is to prevent errors in user input, and to emulate what happens in stores where all this is done automatically

Note on CI/CD: we chose to use CircleCI for CI/CD and had set it up, but since our class CircleCI organization account ran out of credits around Monday in the middle of our development process, we were not able to run CI/CD checks on later commits in our database, and in fact they appeared as “failed” on Github (including the CircleCI passing stickers that we added to our repo readme files). So for the remaining period of time, we turned off branch protection on GitHub, but turned it back on after we finished (it works, we just were not able to run checks and incorporate the CI/CD in our development workflow for later commits due to lacking credits)
For our mobile frontend, we were able to set up CI/CD that ran checks and tests, and could deploy to our production environment (Expo CDN).

CSC301 A1 Report: Jonathan Tang and Ismail Atadinc

We created three repositories for this assignment: one for our mobile frontend in ReactNative (via Expo), one for our web frontend (React), and one for our backend (Django) shared among the mobile and web frontends. We did this so our code would have greater modularity, since with a defined API the codebases are independent of each other, and because this also allows more specific CI/CD scenarios to be set up for each repository.

For the ease of the markers, here are links to our repositories:
Backend Repo
Web Frontend Repo
Mobile Frontend Repo

For our mobile app, we assumed it would only be limited to Android since we don't have access to iOS or Mac devices. Also the Android app ecosystem is more open and accessible than the iOS store: according to our research, the price to publish apps in the Android Play Store is a one-time 25$ whereas Apple Store is 99$/annually.

- Notes at the end of development: we were developing for just Android, but found that because we chose ReactNative via Expo, our code can also run on iOS. It's an added bonus, but not part of our original design nor an official part of our project.

We have put our comparisons of the three technologies in table format, with a brief description of how/why we made our final choice. References to "SO 2020" refer to the 2020 Stack Overflow Developer Survey, looking at responses from professional developers.

Android-Frontend choice

	Android Studio	React Native	Xcode
Ease of development	We have mobile app development experience in it from CSC207	We have no experience	We have no experience in Xcode, Objective-C/Swift, nor the Apple app ecosystem
Maturity & available libraries	Is open source, so it has many libraries	Maturing, has libraries	Has libraries
Domains covered	(Android) App development	Cross-platform app development	(iOS and Mac devices) App development
Popularity	Well known	Growing in popularity	Well known
Performance, scale, and speed	Moderate	Fast	Moderate
Other	Allows access to	Multiplatform	Only officially

considerations	lower-lower native elements, but more time-consuming to create these without templates		supported on MacOS (which we don't have access to at the moment)

We chose React Native because it allows easy porting to iOS, should our hypothetical client like to do that in the future. And it is similar to React, which we chose for web development. Also within React Native, we had to choose Expo CLI or React Native CLI. Our research showed that Expo CLI allows developing for both iOS and Android at the same time compared to React Native CLI which only allows us to develop for one of them at a time. Therefore we chose Expo CLI.

Android-Backend: We considered Java, Python (Django), and Kotlin.

	Java	Python (Django)	Kotlin
Ease of development	We are proficient in Java	We are very proficient in Python	We have no experience
Maturity & available libraries	Very mature language with many libraries	Very mature with many libraries	Growing in maturity
Domains covered	App development	ML, Data science, Scientific computing, Web dev	App development
Popularity	Widely used	Widely used	Not as popular as the other two, but rising
Performance, scale, and speed	Very fast	Fast	Fast
Other considerations		Multiplatform See also notes in Web Backend section	

We chose Python/Django because it can be used to create very portable code. We will also use Python/Django for our web application backend as well so it will reduce the amount of new technology we need to learn in the very short amount of time we have.

Web-Frontend: We considered Angular, React, and Vue.

	Angular	React	Vue

Ease of development	Steep learning curve; uses TypeScript (which we have zero knowledge of)	Quicker to set up; uses JavaScript (which we have some knowledge of)	Simplicity and flexibility leads to high customizability, could be more difficult for us as new beginners working under a tight timeframe. Historically Javascript, but Typescript support added
Maturity & available libraries	Created earliest; mature and has strong backing; continues to evolve today	Mature and has libraries	Growing in maturity, has libraries
Domains covered	Web and mobile development	Web (can develop native mobile apps with React Native, based on React);	Web development
Popularity	Widely used (SO 2020: 27% of professional developers)	Widely used (SO 2020: used by 37% of professional developers)	Not as popular as the other two (SO 2020: 18%), but rising
Performance, scale, and speed	Fast	Fast	Fast
Other considerations	Update every 6 months, one year to accommodate new changes	Has scripts to help migrate between versions	

- We chose React since it is widely used, has a softer learning curve than Angular. It has reusable components for ease of development. Additionally it is also the base for React Native which allows less learning/"initialization" cost for us to learn frameworks for this assignment)

Web-Backend: We considered Django (Python), Node.js, and Flask (Python)

	Django	Node.js	Flask
Ease of development	Uses Python, which is easy to use and both of us are proficient in already	Uses Javascript, which we have some knowledge of	Uses Python, which is easy to use and both of us are proficient in already

Maturity & available libraries	Very mature with many libraries; lots of functionality built-in already	Growing in maturity, has many libraries	Mature with many libraries
Domains covered	ML, Data science, Scientific computing, Web dev	Web dev; has libraries for ML, Data science, Scientific computing	ML, Data science, Scientific computing, Web dev
Popularity	Python is a very convenient and popular language (SO 2020: 4th most common overall language, behind JS, HTML/CSS, SQL), and Django is very popular amongst professionals	Very popular, part of the MEAN stack	Popular with beginners to building websites; same plus of Python
Performance, scale, and speed	Fast	Fast	Fast for small apps
Other considerations	Multiplatform Django ORM allows easy interfacing with databases without knowledge of SQL	Multiplatform The npm package manager comes with Node.js	Multiplatform

    - We considered Django, Node.js, and Flask. We chose Django because it uses Python, a convenient and popular language that we're both much more familiar with. Its "batteries-included" philosophy means it contains many features in the basic framework ready-to-go already, speeding up app development time by reducing configuration time; it also contains the Django ORM for interfacing with databases directly using Python classes and code, which is a plus for our team since we both don't know SQL (and this reduces codebase complexity). Django has a test execution framework, including one that uses Python's unittest.

Android + Web Database: We considered MySQL, PostgreSQL, and MongoDB. (We decided to use the same database backend component for both our mobile and web apps, to reduce code duplication and also have data aggregated in one place)

	MySQL	PostgreSQL	MongoDB
Ease of development	Easy	Easy	Medium

Maturity & available libraries	Mature, has a lot of libraries	Mature, has a lot of libraries	Mature, has a lot of libraries
Domains covered	Machine Learning is easier and more intuitive with relational databases	Machine Learning is easier and more intuitive with relational databases	Machine Learning is harder and less intuitive with non-relational databases
Popularity	Popular (2020 SO: most popular DB)	Popular (2020 SO: second popular DB)	Increasing (2020 SO: most popular NoSQL DB)
Performance, scale, and speed	Fast, SQL better for table-like data; vertically scalable	Fast, SQL better for table-like data; vertically scalable	NoSQL slower for table-like data, better for more flexible database structures; horizontally scalable
Other considerations	Not following SQL standards, so not portable	Following SQL standards, therefore portable	

- We chose PostgreSQL because it is one of the best relational databases. And we want a relational database because it uses table format, which fits our app design best. Note: we ended up using the database that came with Django ORM, which was based on sqlite.

- Android + Web CI/CD: We chose to use the same CI/CD service for our mobile and web apps, to reduce complexity in learning and maintaining the CI/CD services.

	GitLab	Jenkins	Circle CI
Ease of development	We are familiar with GitHub so it will be easier to setup	We have no experience; powerful with many available plugins, but requires setup and configuration	We have no experience, but provides a complete out-of-the-box experience that requires minimal setup
Maturity & available libraries	Mature, many features and users, has libraries. Integration with Github available	Existed for a long time, very mature, many available plugins and extensions, has libraries	Support many languages, including “orbs” that contain reusable configuration elements.

			Integration with Github, Slack available
Domains covered	CI/CD Also some agile/project management integrations	CI/CD	CI/CD
Popularity	Very popular	Popular	Very popular
Performance, scale, and speed	Fast	Moderate	Fast (cached environments and dependencies; free plans have limitations on cloud environment specs)
Other considerations	Github CI/CD does not seem to come with the free version. The cheapest paid tier is 4$ a month, comes with 30 day trial	It is free, open-source, but requires a dedicated server	Cloud-based system, don't have to run server on your own machine; has free plan

We choose CircleCI because it is a popular and convenient cloud-based CI/CD service. For Python and Javascript, it is ready out-of-the-box, and even has reusable configuration elements installed. It also has a GitHub integration that is easy to set up and use. It has a free plan that works.